HS-GCN: A High-performance, Sustainable, and Scalable Chiplet-based Accelerator for Graph Convolutional Network Inference

Yingnan Zhao, *Student Member, IEEE,* Ke Wang, *Member, IEEE,* and Ahmed Louri, *Fellow, IEEE*

***Abstract*—Graph Convolutional Networks (GCNs) have been proposed to extend machine learning techniques for graph-related applications. A typical GCN model consists of multiple layers, each including an aggregation phase, which is communication-intensive, and a combination phase, which is computation-intensive. As the size of real-world graphs increases exponentially, current customized accelerators face challenges in efficiently performing GCN inference due to limited on-chip buffers and other hardware resources for both data computation and communication, which degrades performance and energy efficiency. Additionally, scaling current monolithic designs to address aforementioned challenges will introduce significant cost-effectiveness issues in terms of power, area, and yield. To this end, we propose HS-GCN, a high-performance, sustainable, and scalable chiplet-based accelerator for GCN inference with much-improved energy efficiency. Specifically, HS-GCN integrates multiple reconfigurable chiplets, each of which can be configured to perform the main computations of either the aggregation phase or the combination phase, including Sparse-dense matrix multiplication (SpMM) and General matrix-matrix multiplication (GeMM). HS-GCN implements an active interposer with a flexible interconnection fabric to connect chiplets and other hardware components for efficient data communication. Additionally, HS-GCN introduces two system-level control algorithms that dynamically determine the computation order and corresponding dataflow based on the input graphs and GCN models. These selections are used to further configure the chiplet array and interconnection fabric for much-improved performance and energy efficiency. Evaluation results using real-world graphs demonstrate that HS-GCN achieves significant speedups of 26.7×, 11.2×, 3.9×, 4.7×, 3.1×, along with substantial memory access savings of 94%, 89%, 64%, 85%, 54%, and energy savings of 87%, 84%, 49%, 78%, 41% on average, as compared to HyGCN, AWB-GCN, GCNAX, I-GCN, and SGCN, respectively.**

***Index Terms*—Graph Convolutional Networks, Hardware Accelerators, Dynamic Dataflows, Chiplet-based Design.**

I. Introduction

GRAPH Convolutional Networks (GCNs) have been recently introduced and applied to achieve remarkable inference accuracy in applications using graph-structured data [1]–[7]. A typical GCN architecture comprises multiple graph convolutional layers, and each of these layers consists of two primary computational phases: Aggregation and Combination [8]–[11]. During the aggregation phase, vertices within the input graph collect features from all of their neighbors. In the combination phase, each vertex uses a pre-trained weight matrix to update its aggregated features and generates an output matrix of the current GCN layer. The output matrix of the current GCN layer will be used as an input feature matrix for the next GCN layer. With the explosion of input real-world graphs, the data used for GCN inference becomes extremely tremendous, imposing stringent requirements on data computation and communication when designing hardware accelerators.

Recently, several customized approaches have been proposed to accelerate GCN inference for improved performance [8]–[10], [12], [13]. Although significant achievements, current approaches still face several key limitations when performing GCN inference with large-scale input graphs, which degrades performance and energy efficiency. (1) Due to distinct functionalities, the two phases of GCN inference are either computation-intensive or communication-intensive. However, current monolithic approaches struggle to efficiently execute these phases because of limitations in on-chip computation resources, communication bandwidth, and buffer capacities. [8], [10], [12]. Although scaling current monolithic approaches to provide sufficient hardware resources can achieve better performance, it also introduces significant cost-effectiveness challenges across various aspects, including power, area, and yield [14]–[16]. (2) Current monolithic approaches sequentially execute the GCN phases and layers, thereby failing to exploit and reuse intermediate data across phases and layers. These intermediate data, typically represented as dense matrices generated by matrix-matrix multiplications in each phase, are often overlooked. This neglect results in significant additional on-chip and off-chip memory access, particularly as the size of input graphs increases, ultimately impairing both performance and energy efficiency.

To this end, we propose HS-GCN, a high-performance, sustainable, and scalable chiplet-based accelerator for GCN inference. The main objective of the proposed HS-GCN is to provide flexibility and scalability for GCN inference while reducing data memory access and computations, leading to significantly improved performance and energy efficiency. The main contributions of this work are outlined as follows:

- **A Chiplet-based Architecture:** HS-GCN consists of multiple chiplets, each including a unified computing engine capable of dynamic configuration for efficient execution of key computations in GCN inference. Given an input graph and a specified GCN model, HS-GCN dynamically adapts to the computational and communica-

Yingnan Zhao and Ahmed Louri are with George Washington University, Washington, DC, 20052. Email: {yzhao96, louri}@gwu.edu

Ke Wang is with the University of North Carolina at Charlotte, Charlotte, NC, 28223. E-mail: ke.wang@charlotte.edu.

tion requirements by individually configuring each chiplet as either an aggregation engine or a combination engine. This adaptability enhances performance and improves hardware utilization compared to previous approaches.

- **Flexible Interconnection Fabric:** HS-GCN integrates an active interposer with a flexible interconnection network to establish dynamic connections among chiplets and global buffers with diverse topologies. This provides fast and energy-efficient data transmission for both intra- and inter-chiplet communications. Furthermore, the proposed flexible interconnection allows HS-GCN to efficiently support a wide range of dataflow with diverse on-chip data reuse strategies such as spatial and temporal reuse of the intermediate output tiles, feature tiles, and weight matrices, along with different data reuse patterns.
- **Dynamic Control Algorithms:** HS-GCN introduces two dynamic control algorithms for configuring the proposed reconfigurable chiplets and flexible interconnection fabric, based on the given GCN architecture and input graphs. Specifically, HS-GCN first determines the optimal GCN computation order (the sequence of aggregation and combination phases) based on the given input graph and the applied model, aiming to minimize the number of main memory access throughout the entire inference processing. Subsequently, leveraging the selected computation order, HS-GCN selects an appropriate dataflow to maximize data reuse efficiency across chiplets.

Evaluation results demonstrate that HS-GCN achieves significant speedups, with factors of 26.7×, 11.2×, 3.9×, 4.7×, 3.1×, while also providing substantial memory access savings of 94%, 89%, 64%, 85%, 54%, and energy savings of 87%, 84%, 49%, 78%, 41% on average, as compared to HyGCN, AWB-GCN, GCNAX, I-GCN, and SGCN, respectively.

II. Background, Previous Works, and Motivations

A. GCN Background

Typically, each Graph Convolutional Network (GCN) architecture, also called a GCN model, is composed of multiple graph convolutional layers. Each convolutional layer includes two main computational phases: **Aggregation** and **Combination**. The aggregation phase is generally formulated as Sparse-dense Matrix-matrix Multiplication (SpMM), while the combination phase is General Matrix-matrix Multiplication (GeMM). During the aggregation phase, vertices of the input graph collect feature vectors from their neighbors and aggregate them with the local feature vector. Due to the sparsity of the input graph, each vertex has a different number of neighbors, leading to irregular data memory access and varying numbers of computations. In contrast, during the combination phase, each vertex uses a pre-trained small and dense weight matrix to update the local feature vector [8], [10], [17], [18], which involves regular data memory access and a more consistent number of computations. Subsequently, the updated feature vector of the combination phase will be passed through an activation function, such as the Rectified Linear Unit (ReLU), to obtain the ultimate feature vector, which conducts an output matrix of the current GCN layer. The output matrix then serves as the input feature matrix of the next GCN layer. The main computation of each GCN layer can be abstracted as Eq. 1, which is also shown in Fig. 1 (a).

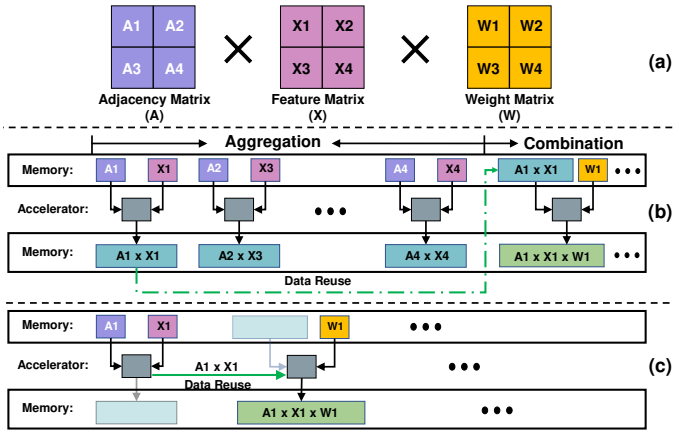


Fig. 1. (a) The adjacency, feature, and weight matrices (A, X, and W) are partitioned into four tiles each. (b) Perform the GCN phases sequentially, processing all tiles within each phase. (c) Reschedule the workload to reuse intermediate results between GCN phases.

$$X^{(k+1)} = \sigma(AX^{(k)}W^{(k)}) \quad (1)$$

Note that the matrix A in Eq. 1 represents the normalized adjacency matrix which is used to record the connection between the source and destination vertices in the input graph. X^k is the feature matrix in layer-k; each row of X denotes a vertex, and each column of X represents a feature. W^k is the well-trained weight matrix of layer-k. $\sigma(.)$ refers to the non-linear activation function such as ReLU [19], [20]. In this paper, HS-GCN mainly focuses on accelerating GCN inference with undirected input graphs.

B. Previous Works

Several customized accelerators have been proposed to accelerate GCN inference and achieve significant improvements compared to general processing units (CPUs and GPUs) [8]–[10], [12]. Specifically, HyGCN [8] implements two separate computing engines to independently perform the aggregation and combination phases of each layer. These two engines are connected in a cascade through an on-chip buffer, where the intermediate data from the aggregation phase is forwarded to the combination engine. AWB-GCN [12] uses a unified computing engine that performs the combination phase first and is designed to address workload imbalances during graph processing. GCNAX [10] tackles resource under-utilization and excessive data movement through a flexible on-chip dataflow. Similarly, I-GCN [9] accelerates GCN execution by merging vertices with shared neighbors, thereby reducing redundant computations and memory access.

C. Motivations

Despite significant achievements, current approaches still face several key limitations that impede further improvements in performance and energy efficiency, particularly as the size of real-world graphs and GCN models grows exponentially.

Limitations in scalability: Since GCN inference is both computation- and communication-intensive, the hardware resource requirements for GCN inference have been increasing exponentially over time, especially as the size of real-world

graphs grows dramatically. Current approaches face challenges in effectively performing GCN inference due to limited on-chip hardware resources, such as the number of computing units, the capacity of on-chip buffers, and on/off-chip bandwidth [15], [16]. Although scaling current monolithic architectures can achieve better performance, it requires a larger die size and further introduces cost-effectiveness challenges in various aspects, including power, area, and yield [14], [15]. Manufacturing an entirely larger chip design with the most advanced technology can cost almost twice as much as using a chiplet approach. For instance, with 16 processing units, a monolithic architecture die (7nm) consumes around 2.5 times more area compared to a chiplet design to achieve similar performance [15]. Therefore, shifting to chiplet-based designs presents a promising solution to efficiently perform GCN inference for large-scale input graphs compared to current approaches.

Limitations in flexibility: Previous works [8], [10], [12] with customized processing elements and interconnection fabric sequentially perform each GCN phase and layer for the entire inference processing, as shown in Fig. 1 (b). However, they failed to provide the necessary flexibility to find and support the data computation and communication requirements induced by the optimal GCN computation order and the corresponding dataflow. For instance, we assume that the dimensions of the feature matrix and weight matrix in Fig. 1 (a) are X $\in R^{NxK}$ and W $\in R^{KxC}$, respectively. If K is smaller than C, using HyGCN [8] will significantly increase the dimension of the inter-phase intermediate matrix from (NxC) to (NxK), which induces extra data memory access. Additionally, with rigid interconnection fabric, current approaches scarify the opportunity to explore the data locality to apply diverse dataflow. For instance, HyGCN [8] and GCNAX [10] only focus on reusing the inter-phase intermediate result without paying attention to the inter-layer intermediate result.

To this end, we propose HS-GCN, a high-performance, sustainable, and scalable chiplet-based accelerator for GCN inference. HS-GCN includes multiple reconfigurable chiplets designed to support computations required by various GCN computation orders. HS-GCN integrates an active interposer with a flexible interconnection fabric, providing the flexibility and scalability needed to meet the data communication requirements of diverse dataflow (data reuse strategies). Additionally, HS-GCN introduces two control algorithms that dynamically select the optimal computation order and dataflow for the given input graph and GCN model.

III. Proposed HS-GCN Architecture

A. Architecture Overview

Fig. 2 depicts the overall architecture of the proposed chiplet-based Graph Convolutional Network (GCN) accelerator, named HS-GCN. HS-GCN comprises an array of reconfigurable chiplets, a flexible interconnection fabric, a control unit, and a global buffer (GLB). The objectives of these proposed components are to provide adaptability, flexibility, and scalability listed in Sec. II-C. Specifically, each chiplet uses a reconfigurable unified computing engine to perform the key computations for GCN inference, including Sparse-dense and General matrix-matrix multiplications (SpMM and GeMM). The design details of the reconfigurable chiplet are discussed in Sec. III-B. The flexible interconnection fabric features an active interposer [21]–[24] with reconfigurable switches and links to establish dynamic connections among chiplets and global buffers through diverse topologies. This provides fast and energy-efficient data transmission for both intra- and inter-chiplet communications. Additionally, the proposed interconnection fabric provides dataflow flexibility by allowing spatial and temporal reuse of all GCN matrices, including adjacency matrix (A), feature matrix (X), weight matrix (W), intermediate data, and final matrix (O). The details of the flexible interconnection fabric are demonstrated in Sec. III-C. The configurations of individual chiplets and the flexible interconnection fabric are supervised by a unified control unit. Given an input graph and a GCN architecture, the control unit has two functions: (1) selecting the optimal GCN computation order and corresponding dataflow strategy for both intra- and inter-chiplet communication and (2) configuring the chiplet array and the interconnection fabric based on these selections, as discussed in Sec. III-D. The global buffer (GLB) is a multi-bank scratchpad connected to the main memory and shared by the chiplet array to store the input, intermediate, and final matrices [25]–[27].

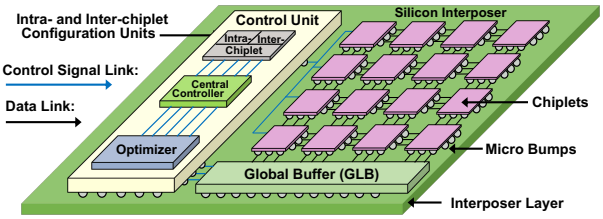


Fig. 2. The overall architecture design of the proposed HS-GCN. The chiplet array comprises N chiplets and is designed to perform the key computations required for GCN inference, including Sparse-dense and General Matrix-matrix Multiplications (SpMM and GeMM). All chiplets are connected through an active interposer with a flexible interconnection network. Both the chiplet array and the interconnection network can be dynamically reconfigured by the control unit to fulfill the requirements of data computation and communication. The control unit is implemented to perform the proposed algorithms that determine the optimal computation order and the configuration of the dataflow strategy applied to both intra- and inter-chiplet. The Global Buffer (GLB) is used to store the input matrices from the main memory, as well as the intermediate and final matrices from the chiplet array.

B. HS-GCN Chiplet Architecture

HS-GCN is comprised of multiple chiplets, each of which is reconfigurable to efficiently perform SpMM and GeMM for GCN inference. As shown in Fig. 3, each chiplet includes a unified computing engine, an Input Sparse Buffer (ISB), an Input Dense Buffer (IDB), a First-In-First-Out (FIFO) unit, a Dense Row Index Unit, an Output Dense Buffer (ODB), and a local controller. Specifically, the unified computing engine comprises a Multiplication engine (MUL Engine) connected to an Accumulation engine (ACC Engine) for required computations. On-chip buffers are used to store different types of input, intermediate, and output data. A local controller is utilized to dynamically configure these hardware components to construct the HS-GCN chiplet as a GeMM engine or an SpMM engine, which are detailed below.

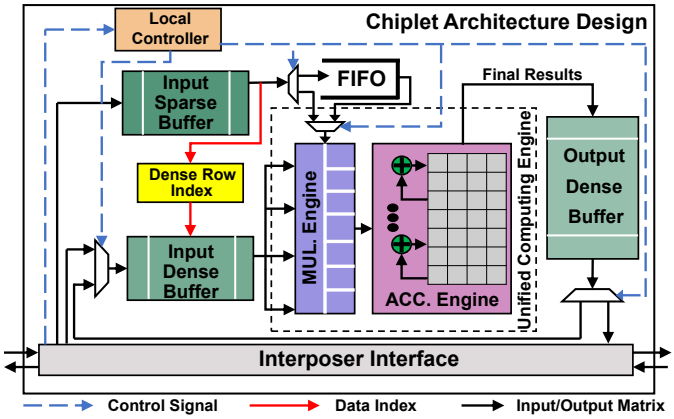


Fig. 3. HS-GCN chiplet architecture overview: The input sparse and dense buffers are used to store input matrices. The output buffer is used to store the intermediate and the final results. The unified computing engine includes a multiplication engine (MUL) and an accumulation engine (ACC) to perform the required matrix-matrix multiplications for both the aggregation and combination phases. The local controller is connected to the global control unit and is used to configure local MUX-DeMUXes to fulfill the data computation and communication requirements.

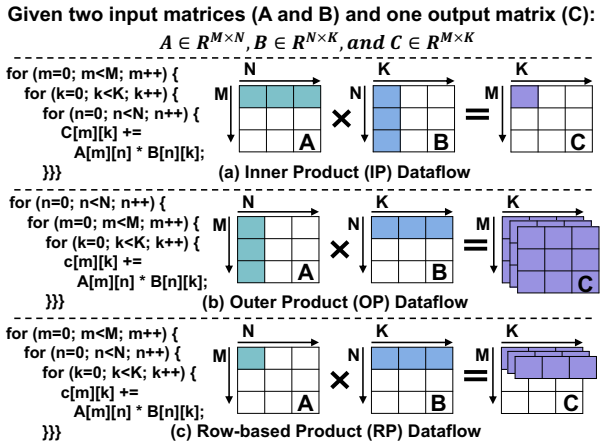


Fig. 4. Three approaches for matrix-matrix multiplications. (a) Inner Product (IP), (b) Outer Product (OP), and (c) Row-based Product (RP).

Configured as GeMM Engine: In this case, the chiplet applies the inner product (IP) to perform required general matrix-matrix multiplications, as shown in Fig. 4 (a). Different from functioning as a SpMM engine, in this mode, the data from the ISB is streamed directly to the MUL engine. Concurrently, the FIFO unit and the Dense Row Index Unit are deactivated. This is because there is no sparsity existing in both input matrices; the indices of data flowing to the computing engine are already corresponding. After receiving data, the MUL and ACC engines perform the element-wise product and partial sum accumulation, respectively. The result of the ACC engine is then stored in the ODB and subsequently transferred to either the interposer interface or the IDB, based on the selected intra-chiplet dataflow strategy, as introduced in Sec. IV-B.

Configured as SpMM Engine: The chiplet applies the row-based product (RP) to perform sparse-dense matrix-matrix multiplication, as shown in Fig. 4 (c). Compared to the outer product (OP), as depicted in Fig. 4 (b), the RP exhibits better efficiency in performing SpMM with a limited on-chip buffer. Specifically, data from the ISB is streamed to the FIFO unit. Concurrently the column index of the data is sent to the Dense Row Index unit. Based on the received column index, the Dense Row Index Unit selects the corresponding row of data from the IDB and transfers the data to the unified computing engine. The FIFO unit and the Dense Row Index Unit work together to ensure that the indices of data streamed from both the IDB and ISB to the unified computing engine match correctly. The unified computing engine works similarly to its functioning in GeMM mode, with the MUL and ACC engines performing multiplications and accumulations, respectively. The ODB is used to receive and store the result locally.

Furthermore, each chiplet integrates a set of dynamic links controlled by MUXes and DeMUXes that allows the input buffers to get data from other chiplets through the interposer as well as directly from the local output buffer. For some dataflow and computation order configurations, this enables direct cross-layer intra-chiplet data reuse to reduce data transmission latency. The details of how this design adapts to diverse dataflow strategies and computation orders will be introduced in Sec. IV-B.

C. The Flexible Interconnection Fabric

HS-GCN implements an active interposer featuring a flexible interconnection network that establishes dynamic connections between chiplets and global buffers. This interconnection fabric facilitates efficient data communication for various dataflows, maximizing data reuse. The proposed interconnection fabric comprises two key hardware components: adaptive interposer switches (IS) and reconfigurable links. Specifically, each IS comprises a switch controller, a dense buffer, a sparse buffer, and pairs of MUX-DeMUXes, as shown in Fig. 5 (a). The switch controller is designed to receive control bits from the global control unit for both intra- and inter-chiplet communication. The intra-chiplet control bit is relayed to the adjacent chiplet, enabling the chiplet to configure the on-chip buffers and the unified computing engine to meet the data computation and communication requirements, as introduced in Sec. III-B. The inter-chiplet control bit is used to configure MUX-DeMUXes within each IS. Depending on the inter-chiplet control bit, each IS dynamically selects input and output ports, activating the corresponding links to establish necessary data paths for communication between chiplets. Since matrices transferred among chiplets can be either sparse or dense, each IS implements two separate local buffers to store the sparse and dense matrices, respectively. By configuring each IS to dynamically establish the connection with its adjacent interposer switches, as shown in Fig. 5 (b), the proposed interconnection fabric provides the flexibility to fulfill the data communication pattern induced by diverse dataflow strategies.

After the configuration of switches is completed, HS-GCN can provide flexible interconnection patterns to manage the data communication, as shown in Fig. 6. Specifically, as shown in Fig. 6 (a) and (b), the flexible interconnection network is configured as mesh-like topologies to support distinct data communication patterns during the GCN's aggregation and combination phases, respectively. During the aggregation

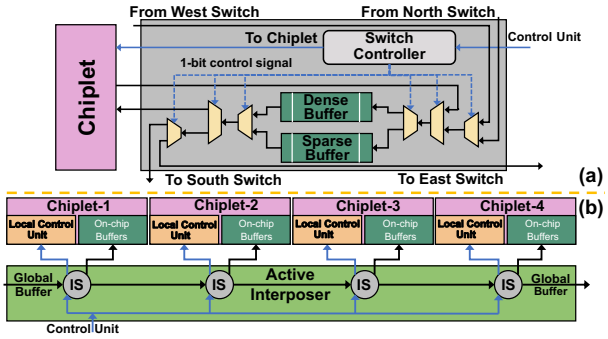


Fig. 5. (a) The detailed architecture of the proposed Interposer Switch (IS) within the interposer layer. Blue links and black links represent the transmission of control signals and data for computation. (b) The interconnection among switches within the interposer layer.

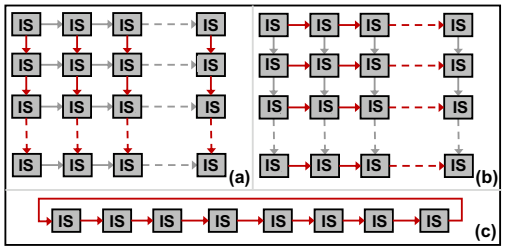


Fig. 6. Three types of reconfigurable interconnections: (a) Each Interposer Switch (IS) only receives data from the north neighbor and transfers the data to the south neighbor. (b) All data is transferred in a horizontal direction. (c) Those switches within the same column are connected in a ring topology.

phase, tiles of the adjacency matrix are horizontally shared among chiplets within the same row, while during the combination phase, tiles of the weight matrix are vertically shared among chiplets involved in the same column. This design choice aligns with the applied tile-based mapping strategy, which assigns different rows of chiplets to different sets of input vertices to mitigate data conflicts during matrix-matrix multiplication and reduce control network complexity. As shown in Fig. 6 (c), the flexible interconnection network can be configured as a ring topology to manage communication between each interposer switch at the edge, primarily designed for those computation orders that consider the reuse of inter-layer intermediate results. To effectively manage data communication among chiplets with minimized control complexity, all chiplets interconnect in a systolic manner. Based on the given tile size, HS-GCN estimates the number of required on-chip computations and data memory accesses for each chiplet as a constant. As a result, all chiplets can simultaneously transmit and receive tiles, thereby preventing potential data asynchronization and conflicts during transmission. When the computation is completed, the final result of each chiplet is sequentially forwarded from left to right, and then to the global buffer for writing back to the main memory. Additionally, the flexible interconnection network can also be configured as a modified torus to maximize inter-layer data reuse.

D. HS-GCN Control Unit

HS-GCN implements a control unit to perform the proposed dynamic control algorithms (Sec. IV) and configures the chiplet array and interposer switches. These algorithms, as detailed in Sec. IV, aim to provide the input graphs with an optimized GCN computation order and corresponding dataflow, thereby improving performance and data reuse efficiency. The control unit comprises three main components: the optimizer, the central controller, and the configuration unit. Specifically, to perform the proposed algorithms, the optimizer retrieves several parameters from the main memory and stores them locally in registers. These parameters include the number of vertices, the length of the feature vector, and the size of the weight matrix. Subsequently, the optimizer uses local Arithmetic Logic Units (ALUs) and the aforementioned parameters to complete the computations required by the proposed algorithms for selecting the computation order. Additionally, the optimizer determines the appropriate tile size from a set of choices based on the selected computation order. The tile size indicates the size of the data chunk loaded from the main memory to the global buffer during each epoch. Once the computation order is selected, the central controller is responsible for determining the corresponding dataflow strategy for both intra- and inter-chiplet communication, aiming to improve data reuse efficiency among chiplets. The determination of the dynamic selection of dataflow strategies is detailed in Sec. IV-A. The configuration unit comprises a small local buffer with multiple entries. The local buffer is implemented to store configuration control bits for each dataflow strategy. When a specific dataflow strategy is selected, the corresponding control bits within the local buffer are sent from the configuration unit to the interconnection network for the management of data communication among different hardware components. Regarding the potential time overhead induced by the control unit, it can be effectively overlapped with the concurrent loading of the feature matrix from main memory into the global buffers. This overlap is feasible because the feature matrix needs to be loaded initially, regardless of whether the aggregation or combination phase is executed first for a given computation order. By prudently overlapping the control operations with the requisite data transfers, the overall execution time impact of the control unit is mitigated.

IV. Proposed Dynamic Control Algorithms

The proposed HS-GCN introduces two dynamic control algorithms, namely a GCN computation order selection algorithm and a dataflow strategy selection algorithm, to achieve improved performance and energy efficiency for any given GCN model and input graph. The selected computation order is used to configure chiplet and reuse intermediate matrices for reduced data memory access. The selected dataflow (including data reuse strategy and data reuse pattern) is used to configure the interconnection network for improved data reuse efficiency between chiplets. In this section, we first provide the details of the two proposed control algorithms, followed by a comprehensive workflow example.

A. Dynamic GCN Computation Order Selection Algorithm

The pseudo-code of a typical GCN computation order is shown in Fig. 7 (a), in which the dimensions of the input adjacency (A), feature (X), and weight (W) matrices are $A \in$

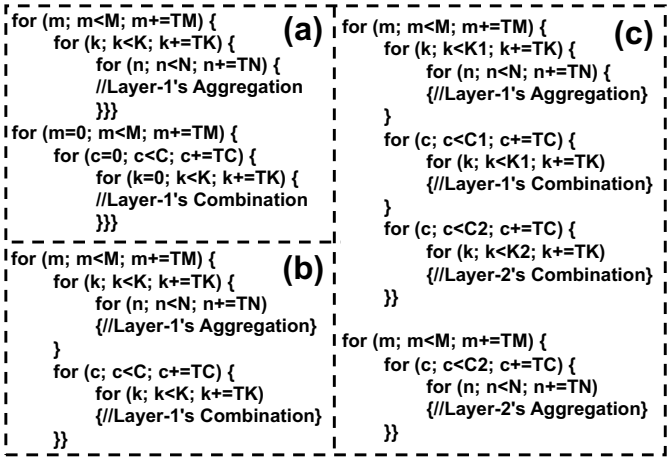


Fig. 7. (a) Without reusing any intermediate result, (b) Reuse the intermediate result of intra-layer, and (c) Reuse the intermediate result of inter-layer.

$R^{(M\times N)}$, $X \in R^{(N\times K)}$, and $W \in R^{(K\times C)}$, respectively. The $< T_M, T_N, T_K, T_C >$ is the tile size tuple which determines the number of data loaded from the main memory to the on-chip buffers during each epoch. Typically, current approaches sequentially perform each GCN layer and store intermediate results in the main memory. However, with the exponential growth in the size of input graphs, current approaches face several challenges. First, the intermediate result of matrix-matrix multiplication becomes large and dense compared to input matrices. Current computation orders without reusing the intermediate results have to frequently access the main memory, which induces additional memory access. Second, for computation orders that reuse inter-layer intermediate results, careful consideration of data dependencies is necessary, especially during two consequent GCN layers. For instance, in a two-layer GCN architecture with the computation order (Layer-1: A(XW) and Layer-2: (AX)W), during the computation of the second GCN layer, each vertex must wait until all its neighbors have updated their features in the first layer to synchronize data. Consequently, the results of the aggregation phase in the first GCN layer cannot be immediately used by the subsequent phase of the second GCN layer.

To address these challenges, we first conduct an extensive design space exploration to identify GCN computation orders that can efficiently reuse intermediate matrices throughout GCN inference. Then, we introduce a dynamic selection algorithm meticulously designed to choose the optimal GCN computation order, prioritizing the reuse of intermediate results while addressing intricate data dependencies inherent in matrix-matrix multiplication for both intra- and inter-layer. The main objective of the proposed selection algorithm is to minimize the data memory access and improve performance.

Since the design space of available GCN computation orders that are capable of reusing intermediate matrices expands exponentially when the number of GCN layers increases, we illustrate our exploration methodology using a 2-layer GCN architecture as an example, as shown in Fig. 7. Typically, the reused intermediate matrices involved in GCN inference can be classified into two types: intra-layer and inter-layer. Fig. 7 (b) and (c) provide details on reusing each type of intermediate matrix, respectively. Fig. 8 lists all available computation orders capable of reusing intermediate results. Due to the data dependencies, several computation orders are excluded from the design space and are not listed in the figure.

Algorithm 1 Dynamic GCN Computation Order Selection Algorithm

1: **Inputs:** The information of input graphs (**M, N, K, S_A, and S_X**). The information of applied GCN architectures (**K and C**). The buffer capacity (**B**). The design space (**S**).
2: **Outputs:** The selected tile size and computation order (**M**).
3: **Begin:**
4: //DRAM access of each execution mode s
5: DA_{s} =0
6: //Record the minimal number of DRAM access
7: DA_min = Integer.MAX_VALUE
8: //Record the mode selection
9: mode = 0
10: **for** k in K **do**
11: T_M = (B+1)/(T_K+1) - 1
12: **for** s in S **do**
13: DA_{s} = Eq.{s}
14: DA_min = min(DA_min, DA_{s})
15: **if** DA_min $\equiv$ DA_{s} **then**
16: //The current mode is selected
17: mode = s
18: **end if**
19: **end for**
20: **end for**
21: **return** mode M

To find the optimal computation order from the design space, the proposed computation order selection algorithm takes multiple parameters of the input graph and the applied GCN architecture. These parameters include the dimensions of the input graph dataset, the applied GCN architecture, the tile size, the buffer capacity, and the design space of available computation orders. Based on a given tile size $< T_M, T_N, T_K, T_C >$, the selection algorithm estimates the number of data memory accesses of each available GCN computation order provided in the design space (Line 12-18). In terms of the estimation function (Line 13), we formulate the data memory access using input parameters such as input graphs and applied models, along with the selected computation order and tile size. Eq. 2 depicts how we estimate the data memory access for a single chiplet implemented in HS-GCN. S_A and S_X represent the sparsity of input adjacency and feature matrices, respectively. We model this during our evaluation to determine the total amount of data required to be loaded from the main memory to the on-chip buffer during each epoch.

$$N = A \times N_{adjacency} + X \times N_{feature} + W \times N_{weight} \quad (2)$$

Where:

$$A = \frac{M \times N}{T_M \times T_N} \qquad N_{adjacency} = T_M \times T_N \times S_A$$
$$X = \frac{N \times K}{T_N \times T_K} \quad (3) \qquad N_{feature} = T_N \times T_K \times S_X \quad (4)$$
$$W = \frac{K \times C}{T_K \times T_C} \qquad N_{weight} = T_K \times T_C$$

After performing the proposed selection algorithm, HS-GCN selects the optimal computation order and tile size pair that leads to the minimum estimated number of data memory

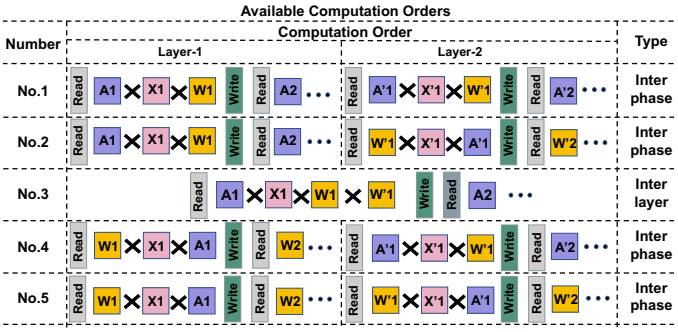


Fig. 8. Available GCN computation orders for reusing intermediate results when considering the data dependency of both intra- and inter-layer.

accesses as the ultimate decision. Subsequently, the selected computation order and the tile size will be applied during the entire inference processing for specific input graphs and the applied GCN architecture. The time complexity of the selection algorithm is $O(m \times n)$, where *m* and *n* represent the choices of available tile size and computation orders. Detailed information is provided in Algorithm 1.

B. Dynamic Dataflow Selection Algorithm

HS-GCN introduces a dynamic dataflow selection algorithm to configure the interconnection network for improved data reuse efficiency among chiplets. Specifically, based on the selected GCN computation order introduced in Sec. IV-A, HS-GCN initially determines whether the intermediate matrix should be reused within a chiplet or among chiplets. With three input matrices (adjacency, feature, and weight matrices), if the decision is to reuse the intermediate matrix within a chiplet, one of the input matrices can be shared among chiplets. Conversely, if the decision is to reuse the intermediate matrix between chiplets, one of the input matrices can stay locally in each chiplet for reuse. Therefore, for a 2-layer GCN architecture, there are a total of 8 ($2 \times 2 \times 2$) candidates within the design space of the dataflow algorithm. The first *2* indicates the number of GCN layers involved in the applied model, the second *2* determines whether the intermediate matrix would be reused among chiplets, and the third *2* indicates if the other two matrices except the intermediate matrix are reused during GCN processing. Given the design space, the proposed algorithm takes several parameters, including the GCN computation order, the tile size, and the number of chiplets, to estimate the number of data memory access for each available dataflow, as detailed in Algorithm 2.

For the estimation function used in Algorithm 2, when provided with two input matrices and one output matrix, one of these three matrices remains locally within each chiplet for potential reuse, while the other two are transmitted to neighboring chiplets for sharing. Consequently, we formulate the data movement aspect of the estimation to assess the traffic between neighboring chiplets when sharing data, as shown in Algorithm 3. After the optimal dataflow is selected to maximize data movement efficiency, HS-GCN uses the chosen dataflow to configure both the interconnection network and the chiplet array to establish the data communication path. Additionally, HS-GCN applies a mapping strategy that assigns different vertices to chiplets located on different rows

Algorithm 2 Dynamic Dataflow Selection Algorithm

Inputs: The dimension of accelerators (**R**×**C**). The determined tile size (T_M, T_N, T_K, T_C). The computation order (**mode M**). Available dataflow set (**S**)
Outputs: The selected dataflow decision (**D**).
Begin:
//Record the data movement
DM_min = 0
//Data movement of each scheduling selection
DM_{s} =0
//Record the minimal number of DRAM access
DM_min = Integer.MAX_VALUE
//Record the final decision
decision = 0
for s in S **do**
    DM_{s} = count_DM(T_M, T_N, T_K, T_C, R, C, s)
    if DM_min ≡ DM_{s} **then**
        //The current scheduling is selected
        decision = s
    end if
end for
return decision D

Algorithm 3 Data Movement Counting Function (count_DM)

Inputs: The dimension of accelerators (**R**×**C**). The determined tile size (T_M, T_N, T_K, T_C). The current dataflow selection (**s**)
Outputs: The number of data movement DM.
Begin:
//Assume the adjacency matrix is currently reused
DM_reuse = T_M × T_N × R × C
DM_d1 = T_M × T_K × (R+1) × C
DM_d2 = T_N × T_K × R × (C+1)
DM_total = DM_reuse + DM_d1 + DM_d2
return DM_total

and assigns different features of the same vertex to chiplets located on different columns. This mapping approach serves a dual purpose: it helps avoid conflicts during data transmission and concurrently ensures data synchronization during matrix-matrix multiplication.

C. Workflow Example

For clarity, let's illustrate the proposed architecture using an example comprised of n×n chiplets. We provide a detailed description of the workflow of the proposed approach, as shown in Fig. 9, where n is 4 in the figure. The three input matrices are as follows: $A \in R^{(M \times N)}$, $X \in R^{(N \times K)}$, and $W \in R^{(K \times C)}$. This implies that there are N vertices in the input graph, with each vertex's feature vector having a length of K. The tile size is $< T_M, T_N, T_K, T_C >$ after performing the proposed algorithm introduced in Sec. IV-A. Additionally, as introduced in Sec. IV-A, we adopt the last computation order (No. 5) as the computation order for this example. Once the computation order and dataflow have been determined, the features of vertices are fetched from the main memory to the GLB and subsequently to the designated chiplet, where they are stored locally before performing the computations (Step #0). The details of the workflow are as follows:

Step #1: All chiplets read their designated tiles of the weight matrix (W) from the GLB through the interconnection and store the data inside the local buffer, as shown in Fig. 9 (b). In this specific example, as the dimension of the tile size matches

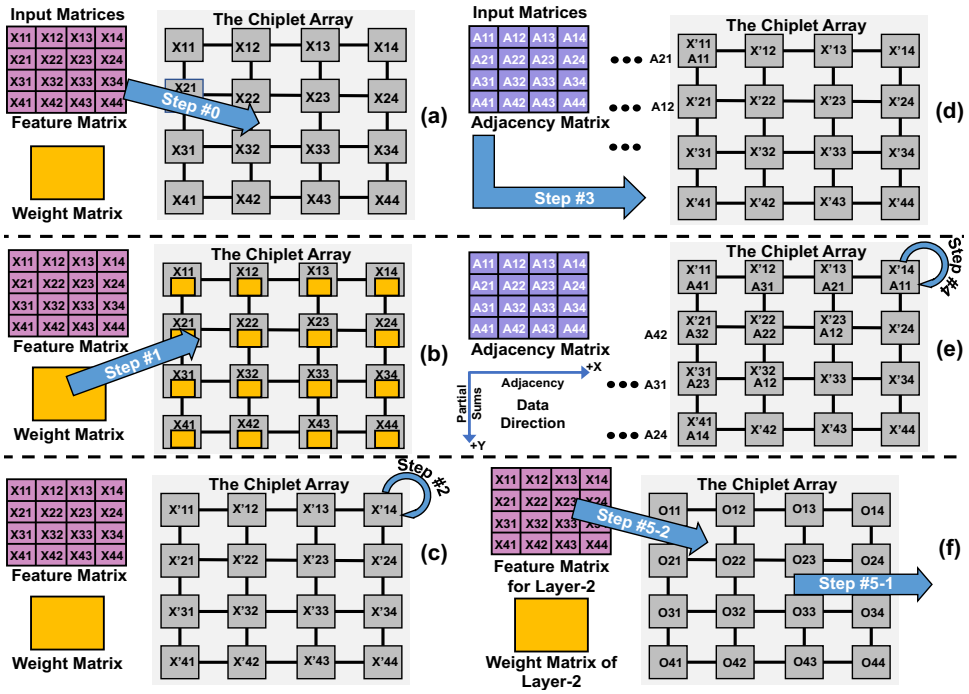


Fig. 9. Example Workflow: Introducing the proposed architecture design with three small-scale GCN input matrices (A, X, and W) and a n×n chiplet array, where n is 4 in this figure. The workflow mainly includes 5 steps in total.

that of the weight matrix, it is possible to store the entire weight matrix in each chiplet. In actual GCN architectures, the dimensions of the weight matrix are typically small as well [8], [10].

Step #2: Following the loading of the weight matrix, each chiplet activates its local unified computing engine, utilizing the stored tiles of feature and weight matrices to perform computations for the combination phase. The resulting updated tiles of features, which are intermediate results, are retained locally within the output dense buffer, as shown in Fig. 9 (c).

Step #3: In this step, the architecture encompasses two primary operations. Firstly, the data, which consists of intermediate results stored within the output dense buffer of each chiplet, is forwarded to the local input dense buffer to serve as input for subsequent computations through the reconfigurable DeMUX. Secondly, tiles of the adjacency matrix from the GLB are sent to the chiplet array in a systolic manner through the interconnection and stored inside the input sparse buffer. The aforementioned operations are executed concurrently, as shown in Fig. 9 (d).

Step #4: During this step, the architecture also involves two primary operations. Firstly, after the adjacency matrix is loaded, each chiplet utilizes the stored tiles of intermediate results and the adjacency matrix to perform computations for the aggregation phase through the unified computing engine, as illustrated in Fig. 9 (e). Subsequently, the chiplet will transmit the adjacency matrix to its neighboring chiplet within the same row. This communication is facilitated through those interposer switches (IS) located in the interposer layer as introduced in Sec. III-C. The aggregation phase for the current tiles concludes once the intermediate results have traversed all the chiplets within the same row.

Step #5: Once completing the required computations, the intermediate results contained within the output dense buffer of each chiplet are streamed out to the GLB through the DeMUX and subsequently to the main memory, as shown in Fig.9 (f). Concurrently, the architecture initiates the loading of the next batch of tiles of matrix X into the respective chiplets for further computations. Additionally, these two types of data are transmitted through two distinct topologies to avoid data conflicts, as explained in Sec. III-C.

(a) Chiplet Layout Characteristics

Component	Size
MUL. Engine	16 x 1 multipliers
ACC. Engine	15 adders
Input Sparse Buffer	128 KB
Input Dense Buffer	4 KB
Output Dense Buffer	256 KB
Local Control Unit	-

(b) Overall Layout Characteristics

Component	Size
Chiplet Array	4 x 4
Switches Array	4 x 4
Global Buffer (GLB)	1 MB
Optimizer	10 MAC units
	8 registers
Central Controller	-
Bandwidth	256 GB/s

Fig. 10. (a) Layout characteristics of each chiplet. (b) Layout characteristics of the overall architecture.

V. Evaluation Methodologies

A. Evaluation Setup

Hardware simulator. We build a cycle-accurate simulator in C++ language to evaluate the hardware behavior and the performance of the proposed design. Specifically, the simulator precisely counts the exact number of memory read and write operations, which is used to estimate the energy consumption of the memory access according to [28]. To measure the area consumption, we implement all the proposed hardware logic, including the chiplet design, the interposer switch design, and other hardware components through the Synopsys Design Compiler with the TSMC 45nm library for the synthesis. We set the clock frequency at 1 GHz. We use Cacti 6.0 [29] and

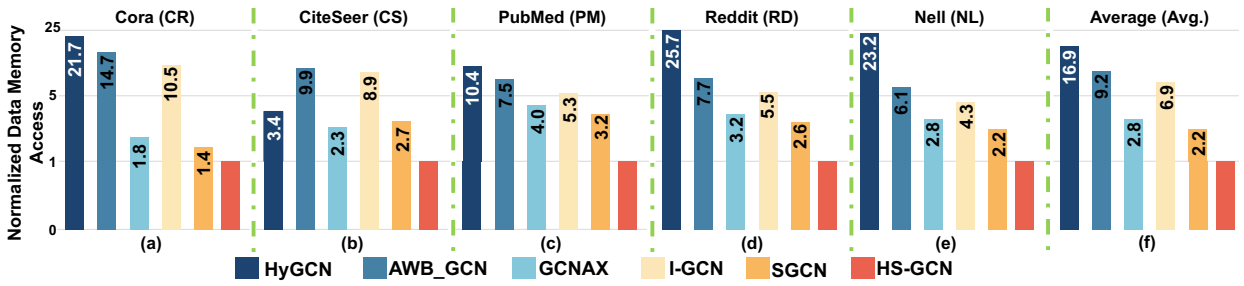


Fig. 11. The normalized data memory access of the proposed design (HS-GCN) compared to prior accelerators. (a) Cora (CR), (b) CiteSeer (CS), (c) PubMed (PM), (d) Nell (NL), (e) Reddit (RD), and (f) Average (Avg.) (lower is better).

DSENT [30] to estimate the area, power, and access latency of all types of buffers and links.

Architecture Configuration. Fig. 10 (a) and Fig. 10 (b) list the layout characteristics of each chiplet design and the overall architecture. Specifically, the proposed design implements an N×N chiplet array with K×K interposer switches, where both N and K equal 4 during our evaluation. Additionally, in Sec. VI-D, we conduct a scalability analysis of the proposed design to illustrate the relationship between the execution time and the dimensions of the unified chiplet array. Each Chiplet is equipped with 16×1 multipliers cascaded 15 adders in a binary tree architecture to perform the necessary matrix-matrix multiplication required for GCN inference. For the optimizer, it includes 8 registers to receive and store the parameters of the input application, and 10 MAC units to perform the required computations required by the proposed algorithms. The sizes of all on-chip buffers are designed with sufficient capacity to store the data required for computations. For instance, the GLB serves as a bridge facilitating data communication between the main memory and the chiplet array. As connecting to the first column of the array, the GLB is sized at 1MB to ensure sufficient capacity for the data volume being transferred.

Baselines. We compare the proposed design to five previous customized accelerators (HyGCN [8], AWB-GCN [12], GCNAX [10], I-GCN [9], and SGCN [13]) using the same simulator platform to ensure a fair comparison. Given that the proposed design features a 4 x 4 chiplet array, with each chiplet comprising 16 multipliers, the total number of multipliers amounts to 256. Consequently, all baseline accelerators have been scaled to incorporate an equivalent number of computation units as the proposed design. A previous study [8] has shown that utilizing single-precision floating-point numbers (4 Bytes per data) is sufficient to preserve GCN inference accuracy. As a result, both computing units and links have a width of 32 bits for data computation and communication in the majority of accelerators. However, GCNAX differs in its regard, as it utilizes 64-bit wide computing units and links due to its operation with double precision (8 Bytes per data). For a fair comparison, the main memory bandwidth for all accelerators is scaled to 256GB/s. Since the aforementioned customized accelerators outperform general-purpose CPUs and GPUs, we did not include comparisons against those general processing units.

Evaluation Datasets. In this paper, we leverage commonly used datasets from previous literature [3], [31]–[35] to conduct the further experimental evaluation. These datasets include Cora (CR), CiteSeer (CS), PubMed (PM), Nell (NL), and Reddit (RD). Cora, CiteSeer, and PubMed are well-known datasets for paper citation networks, node classification, and text summarization [1], [33]. The Reddit dataset represents an undirected graph of social networks, comprising posts gathered from the Reddit discussion forum. The Nell dataset, on the other hand, is a knowledge graph obtained from the Never-Ending Language Learning project. Table V-A provides detailed information about each dataset used in this study, including its structure and data density. Additionally, the last column of the table provides information about the change in the feature vector's length during the entire GCN inference process. Obviously, except for the NELL dataset, the length of feature vectors in most graph datasets decreases.

TABLE I
DETAILS OF GCN DATASETS USED FOR EVALUATION

Datasets	Vertices	Edges	Sparsity	Feature Length
Cora (CR)	2708	10556	0.018%	1433-16-7
CiteSeer (CS)	3327	9104	0.11%	3703-16-6
PubMed (PM)	1917	88648	0.028%	500-16-3
Nell (NL)	65755	266144	0.0073%	61278-64-186
Reddit (RD)	232965	114615892	0.21%	602-64-41

VI. Evaluation and Analysis

A. Data Memory Access

Fig. 11 shows the normalized data memory access of the proposed design compared to previous works. The proposed design outperforms the previous approaches for the following reasons: Firstly, HS-GCN selects the optimal GCN computation order while considering the data reuse of intermediate results for diverse input graphs. Consequently, HS-GCN overlaps the execution of consecutive phases, ensuring that the large and dense intermediate results can be immediately utilized by the subsequent GCN phase and layer without the need to store and load them to and from the main memory repeatedly, thereby reducing the total data memory accesses. Secondly, HS-GCN offers a flexible dataflow along with a reconfigurable interconnection network that allows the chiplet to retrieve the necessary data from its neighboring chiplet instead of the main memory. This improvement enhances the overall data reuse efficiency. With all data being normalized to the proposed design, as shown in Fig. 11, the proposed architecture provides a memory access reduction by a factor of 16.9×, 9.2×, 2.8×, 6.9×, 2.2×, which imply a memory access reduction by 94%, 89%, 64%, 85%, 54% compared to HyGCN, AWB-GCN, GCNAX, I-GCN, and SGCN, respectively.

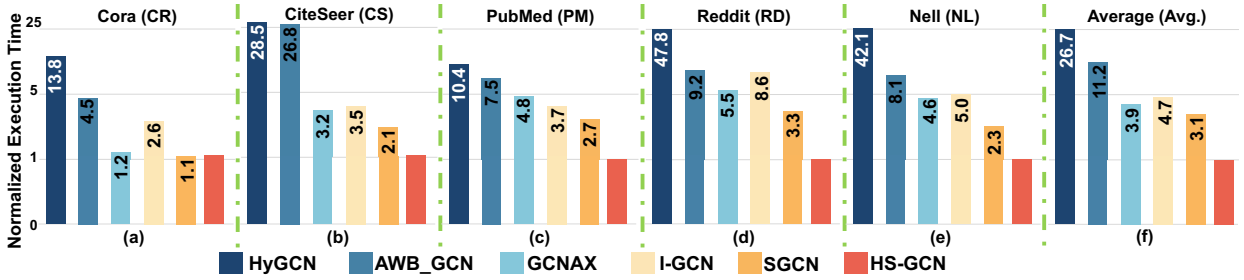


Fig. 12. The normalized execution time (cycles) of the proposed design (HS-GCN) compared to prior accelerators. (a) Cora (CR), (b) CiteSeer (CS), (c) PubMed (PM), (d) Nell (NL), (e) Reddit (RD), and (f) Average (Avg.) (lower is better).

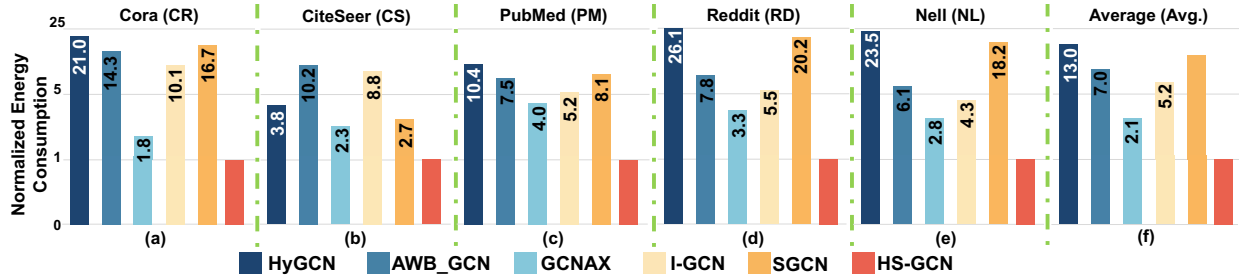


Fig. 13. The normalized energy consumption of the proposed design (HS-GCN) compared to prior accelerators. (a) Cora (CR), (b) CiteSeer (CS), (c) PubMed (PM), (d) Nell (NL), (e) Reddit (RD), and (f) Average (Avg.) (lower is better).

B. Execution Time

Fig. 12 illustrates the normalized execution time of the proposed HS-GCN compared to previous works. HS-GCN achieves an execution time speedup of a factor of 26.7×, 11.2×, 3.9×, 4.7×, 3.1× on average of real-world GCN datasets, compared to HyGCN, AWB-GCN, GCNAX, I-GCN, and SGCN, respectively. In contrast to existing approaches, HS-GCN uses a novel control algorithm coupled with a flexible interconnection fabric to enhance data reuse efficiency, thereby reducing memory access, which is a key performance bottleneck. For example, when processing the NELL (NL) dataset, which has large and dense intermediate data across phases and layers, HS-GCN effectively reuses these data within the chiplet array. This approach eliminates the need for frequent data transfers between memory and computation units, thereby improving overall performance. The time overhead associated with executing the control algorithm and performing the configuration setup can be effectively overlapped. Since all types of computation orders require the feature matrix at the beginning of inference, the system concurrently loads the feature matrix of input graphs from the main memory to the global buffer while the control unit executes the algorithms and configures the chiplet array and the interconnection fabric.

C. Energy Consumption

All accelerators estimate the related energy consumption according to [29], and all values are normalized based on the proposed HS-GCN design. As shown in Fig. 13, HS-GCN achieves 13.0×, 7.0×, 2.1×, 5.2×, 1.8× energy savings, which imply 87%, 84%, 49%, 78%, 41% energy reduction on average compared to HyGCN, AWB-GCN, GCNAX, I-GCN, and SGCN, respectively. Since memory access plays a major bottleneck for energy consumption, HS-GCN uses the proposed control algorithms in conjunction with the interconnection fabric as the primary approach to significantly

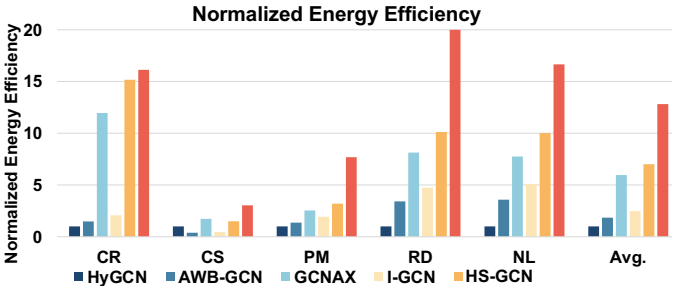


Fig. 14. The energy efficiency of HS-GCN compared to previous works across various input graph datasets, including Cora (CR), CiteSeer (CS), PubMed (PM), Reddit (RD), Nell (NL), and the average (Avg.). All evaluation results are normalized to the performance of HyGCN (higher is better).

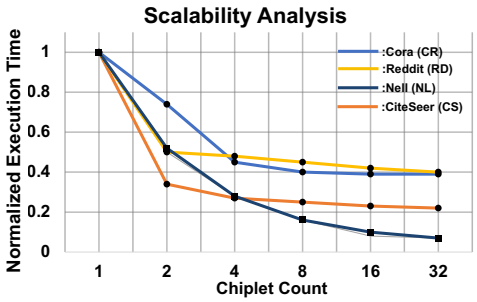


Fig. 15. The scalability analysis of the proposed design (HS-GCN) with various graph datasets. All evaluation results are normalized based on using a single chiplet with optimal computation order and dataflow.

reduce memory access, as detailed in Sec. VI-A. Specifically, HS-GCN first optimizes the GCN computation order to minimize overall memory access. Subsequently, based on the selected computation order, HS-GCN determines an appropriate dataflow to maximize data reuse efficiency, further reducing memory access. To accommodate the diverse matrix-matrix multiplication resulting from different computation orders, HS-GCN integrates a reconfigurable chiplet design to meet data computation requirements efficiently.

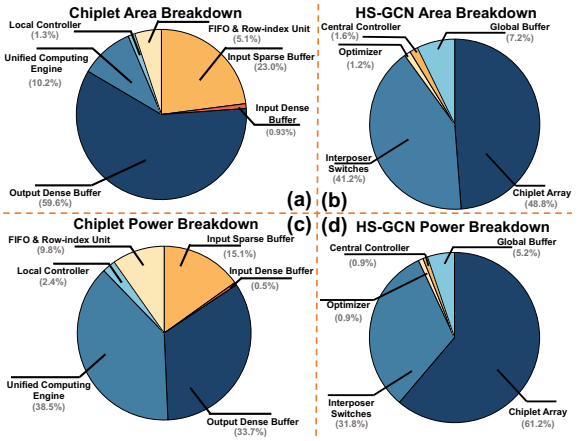


Fig. 16. (a) and (b) represent the area breakdown for each chiplet and the proposed design (HS-GCN). (c) and (d) illustrate the power breakdown for each chiplet and the proposed design (HS-GCN).

D. Scalability Analysis

Fig. 15 illustrates the impact on execution time when scaling HS-GCN across various datasets, ranging from 1 to 32 chiplets. Regarding the scaling process, using 8 chiplets as an example, we estimate the execution time for both (2×4) and (4×2) arrays, as these two configurations exhibit differing performances. Subsequently, we calculate the average value of two distinct results to determine the overall execution time. As the number of chiplets increases, the overall execution time is significantly reduced across all evaluated datasets due to the availability of sufficient hardware resources to fulfill the computation and communication requirements during DGCN inference. To mitigate the latency of data communication between chiplets as their number increases, HS-GCN uses the proposed algorithm to predetermine the tile size and ensure that the data chunks transferred between adjacent chiplets remain consistent. Consequently, even as the number of chiplets increases, the latency for each chiplet to fetch data from its neighbors remains relatively uniform. As substantial computing resources are provided for input graphs, the primary bottleneck shifts to the bandwidth between the main memory and the global buffer, as indicated by the Roofline model. Therefore, this shows that increasing on-chip resources is unable to provide a proportional reduction in execution time.

E. Area and Power Analysis

Fig. 16 (a) and (b) provide a comprehensive breakdown of the area consumption for both the chiplet and the proposed HS-GCN. For each chiplet, on-chip buffers occupy the majority of the area, accounting for approximately 82% of the total. Since the proposed HS-GCN is a chiplet-based architecture, the area consumption of the chiplet array can be overlapped by part of the interposer layer. Fig. 16 (c) and Fig. 16 (d) show a detailed power breakdown for both the chiplet and the HS-GCN. Notably, the chiplet array and the interposer switches are the primary hardware components that consume the majority of power, accounting for approximately 90% of the total.

VII. Conclusion

This paper proposes HS-GCN, a chiplet-based GCN accelerator designed to address the critical limitations of current approaches, including scalability and flexibility. HS-GCN comprises multiple chiplets designed to efficiently perform the primary computations involved in GCN inference. Furthermore, HS-GCN integrates an active interposer with a flexible interconnection fabric to accommodate diverse dataflows and their associated communication patterns. In addition, HS-GCN introduces algorithms to determine optimal GCN computation orders and dataflows, enabling the reuse of intermediate matrices and the dynamic configuration of both the chiplet array and the interconnection fabric.

References

[1] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[2] Tao Liu, Peng Li, Zhou Su, and Mianxiong Dong. Efficient inference of graph neural networks using local sensitive hash. *in IEEE Transactions on Sustainable Computing*, January, 2024.

[3] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *in IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, March, 2020.

[4] Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *in IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 1, pp. 249–210, March, 2020.

[5] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *in AI open*, vol. 1, pp. 57–81, January, 2020.

[6] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *in IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, December, 2008.

[7] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[8] Mingyu Yan, Lei Deng, Xing Hu, Ling Liang, Yujing Feng, Xiaochun Ye, Zhimin Zhang, Dongrui Fan, and Yuan Xie. Hygcn: A gcn accelerator with hybrid architecture. In *Proceedings of 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 1–14. San Diego, CA, USA, February 22-26, 2020.

[9] Tong Geng, Chunshu Wu, Yongan Zhang, Cheng Tan, Chenhao Xie, Haoran You, Martin Herbordt, Yingyan Lin, and Ang Li. I-gcn: A graph convolutional network accelerator with runtime locality enhancement through islandization. In *Proceedings of 2021 Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1051–1063. Athens, Greece, October 18-22, 2021.

[10] Jiajun Li, Ahmed Louri, Avinash Karanth, and Razvan Bunescu. Gcnax: A flexible and energy-efficient accelerator for graph convolutional neural networks. In *Proceedings of 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1–14. Seoul, South Korea, February 27-March 3, 2021.

[11] Yingnan Zhao, Ke Wang, and Ahmed Louri. Opt-gcn: A unified and scalable chiplet-based accelerator for high-performance and energy-efficient gcn computation. *in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, May, 2024.

[12] Tong Geng, Ang Li, Runbin Shi, Chunshu Wu, Tianqi Wang, Yanfei Li, Pouya Haghi, Antonino Tumeo, Shuai Che, Steve Reinhardt, et al. Awb-gcn: A graph convolutional network accelerator with runtime workload rebalancing. In *Proceedings of 2020 Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–14. Athens, Greece, October 17-21, 2020.

[13] Mingi Yoo, Jaeyong Song, Jounghoo Lee, Namhyung Kim, Youngsok Kim, and Jinho Lee. Sgcn: Exploiting compressed-sparse features in deep graph convolutional network accelerators. In *Proceedings of 2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1–14. Montreal, QC, Canada, February 25-March 01, 2023.

[14] Yakun Sophia Shao, Jason Clemons, Rangharajan Venkatesan, Brian Zimmer, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, et al. Simba: Scaling deep-learning inference with multi-chip-module-based architecture. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 14–27. Columbus, OH, USA, October 12-16, 2019.

[15] Samuel Naffziger, Noah Beck, Thomas Burd, Kevin Lepak, Gabriel H Loh, Mahesh Subramony, and Sean White. Pioneering chiplet technology and design for the amd epyc™ and ryzen™ processor families: Industrial product. In *Proceedings of the 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 57–70. Virtual, June 14-19, 2021.

[16] Patrick Iff, Maciej Besta, Matheus Cavalcante, Tim Fischer, Luca Benini, and Torsten Hoefler. Hexamesh: Scaling to hundreds of chiplets with an optimized chiplet arrangement. In *Proceedings of the 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. San Francisco, CA, USA, June 23-27, 2023.

[17] Yanhong Wang, Tianchan Guan, Dimin Niu, Qiaosha Zou, Hongzhong Zheng, C-J Richard Shi, and Yuan Xie. Accelerating distributed gnn training by codes. *in IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 34, no. 9, pp. 2598–2614, July, 2023.

[18] Jiajun Li, Hao Zheng, Ke Wang, and Ahmed Louri. Sgcnax: A scalable graph convolutional neural network accelerator with workload balancing. *in IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 33, no. 11, pp. 2834–2845, December, 2021.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *in Advances in Neural Information Processing Systems*, 25, 2012.

[20] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[21] Hao Zheng, Ke Wang, and Ahmed Louri. Adapt-noc: A flexible network-on-chip design for heterogeneous manycore architectures. In *Proceedings of 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 723–735. Seoul, South Korea, February 27-March 03, 2021.

[22] Natalie Enright Jerger, Ajaykumar Kannan, Zimo Li, and Gabriel H Loh. Noc architectures for silicon interposer systems: Why pay for more wires when you can get them (from your interposer) for free? In *Proceedings of 2014 IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 458–470. Cambridge, UK, December 13-17, 2014.

[23] Ajaykumar Kannan, Natalie Enright Jerger, and Gabriel H Loh. Enabling interposer-based disintegration of multi-core processors. In *Proceedings of 2015 IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 546–558. Waikiki, Hawaii, USA, December 5-9, 2015.

[24] Dylan Stow, Yuan Xie, Taniya Siddiqua, and Gabriel H Loh. Cost-effective design of scalable high-performance systems using active and passive interposers. In *Proceedings of 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 728–735. Irvine, California, USA, November 13-17, 2017.

[25] Xinkai Song, Tian Zhi, Zhe Fan, Zhenxing Zhang, Xi Zeng, Wei Li, Xing Hu, Zidong Du, Qi Guo, and Yunji Chen. Cambricon-g: A polyvalent energy-efficient accelerator for dynamic graph neural networks. *in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 41, no. 1, pp. 116–128, January, 2021.

[26] Tae Jun Ham, Lisa Wu, Narayanan Sundaram, Nadathur Satish, and Margaret Martonosi. Graphicionado: A high-performance and energy-efficient accelerator for graph analytics. In *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–14. Taipei, Taiwan, October 15-19, 2016.

[27] Priyank Faldu, Jeff Diamond, and Boris Grot. Domain-specialized cache management for graph analytics. In *Proceedings of the 27th IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1–14. San Diego, CA, USA, February 22-26, 2020.

[28] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: Efficient inference engine on compressed deep neural network. *in ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.

[29] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P Jouppi. Cacti 6.0: A tool to understand large caches. *University of Utah and Hewlett Packard Laboratories, Tech. Rep*, 147, 2009.

[30] Chen Sun, Chia-Hsin Owen Chen, George Kurian, Lan Wei, Jason Miller, Anant Agarwal, Li-Shiuan Peh, and Vladimir Stojanovic. DSENT-a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. In *Proc. of NOCS'12*, pages 201–210. IEEE, 2012.

[31] Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and S Yu Philip. A survey on knowledge graphs: Representation, acquisition, and applications. *in IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 2, pp. 494–514, April, 2021.

[32] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *in Information Retrieval*, vol. 3, pp. 127–163, July, 2000.

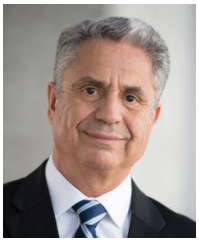
[33] C Lee Giles, Kurt D Bollacker, and Steve Lawrence. Citeseer: An automatic citation indexing system. In *Proceedings of the 3rd ACM Conference on Digital Libraries*, pages 89–98. Pittsburgh, PA, USA, June, 1998.

[34] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *in AI magazine*, vol. 29, no. 3, pp. 93–93, September, 2008.

[35] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st Annual Conference in Neural Information Processing Systems (NeurIPS)*. Long Beach, CA, USA, December 4-9, 2017.

Yingnan Zhao received the BS degree in computer science from the Zhejiang University of Technology, China, in 2018, and the MS degree in electrical engineering from the George Washington University, US, in 2020. He is currently working toward a Ph.D. degree in computer engineering at George Washington University, Washington, DC. His research interests include graph neural networks, AI accelerator design, and interconnection networks.

Ke Wang received the Ph.D. degree in Computer Engineering from the George Washington University in 2022. He received the M.S. degree in Electrical Engineering from Worcester Polytechnic Institute in 2015, and the B.S. degree in Electrical Engineering from Peking University in 2013. He is currently an Assistant Professor of Electrical and Computer Engineering at the University of North Carolina at Charlotte. His research work focuses on parallel computing, computer architecture, interconnection networks, and machine learning.



Ahmed Louri (Fellow, IEEE) received the PhD degree in computer engineering from the University of Southern California, Los Angeles, California, in 1988. He is the David and Marilyn Karlgaard Endowed chair professor of electrical and computer engineering at the George Washington University, Washington, DC., which he joined in August 2015. He is also the director of High Performance Computing Architectures and Technologies Laboratory. From 1988 to 2015, he was a professor of electrical and computer engineering at the University of Arizona, Tucson, Arizona, and during that time, he served six years (2000 to 2006) as the chair of the Computer Engineering Program. From 2010 to 2013, he served as a program director in the National Science Foundation's (NSF) Directorate for Computer and Information Science and Engineering. He directed the core computer architecture program and was on the management team of several cross-cutting programs. He conducts research in the broad area of computer architecture and parallel computing, with emphasis on interconnection networks, optical interconnects for parallel computing systems, reconfigurable computing systems, and power-efficient and reliable Network-on-Chips (NoCs) for multicore architectures. Recently he has been concentrating on energy-efficient, reliable, and high-performance many-core architectures, accelerator-rich reconfigurable heterogeneous architectures, machine learning techniques for efficient computing, memory, and interconnect systems, emerging interconnect technologies (photonic, wireless, RF, hybrid) for NoCs, future parallel computing models and architectures (including convolutional neural networks, deep neural networks, and approximate computing), and cloud-computing and data centers. He is the recipient of 2020 IEEE Computer Society Edward J. McCluskey Technical Achievement Award for pioneering contributions to the solution of on-chip and off-chip communication problems for parallel computing and many-core architectures. For more information, please visit https://hpcat.seas. gwu.edu/Director.html.